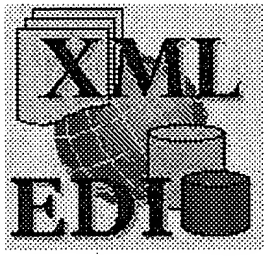
L Number	Hits	Search Text	DB	Time stamp
1	86	@ad < "20011127" and (retransmit same (transmission adj2 error)) and network	USPAT; US-PGPUB	2004/03/02 16:21
2	11	@ad < "20011127" and (retransmit same (transmission adj2 error)) and network and "error message"	USPAT; US-PGPUB	2004/03/02 16:21
-	124	@pd < "20011127" and EDI and schedule	USPAT; US-PGPUB	2004/03/02 08:13
-	6	@pd < "20011127" and EDI same schedule	USPAT; US-PGPUB	2004/03/02 08:13
-	1	@pd < "20011127" and EDI same schedule and XML	USPAT; US-PGPUB	2004/03/02 08:13
-	1	@pd < "20011127" and (EDI same schedule) and XML	USPAT; US-PGPUB	2004/03/02 08:59
-	3	@pd < "20011127" and (load same XML same database)	USPAT; US-PGPUB	2004/03/02 09:00
-	1	@pd < "20011127" and (loader same XML same database)	USPAT; US-PGPUB	2004/03/02 09:01
-	2	@pd < "20011127" and (loads same XML same database)	USPAT; US-PGPUB	2004/03/02 09:01
-	1	@pd < "20011127" and (import$2 same XML same database)	USPAT; US-PGPUB	2004/03/02 09:02
-	41	@ad < "20011127" and microsoft.asn. and (XML same database)	USPAT; US-PGPUB	2004/03/02 09:03
-	2	@ad < "20011127" and microsoft.asn. and ((load$2 or import$2) same XML same database)	USPAT; US-PGPUB	2004/03/02 09:03
-	176	@ad < "20011127" and ((load$2 or import$2) same XML same database)	USPAT; US-PGPUB	2004/03/02 09:03
-	9	@ad < "20011127" and ((load$2 or import$2) adj2 XML adj4 database)	USPAT; US-PGPUB	2004/03/02 16:20
-	75	@pd < "20011127" and (EDI same database)	USPAT; US-PGPUB	2004/03/02 09:07
-	0	@pd < "20011127" and (EDI same schedule same database)	USPAT; US-PGPUB	2004/03/02 09:07
-	6	@pd < "20011127" and (EDI same schedule)	USPAT; US-PGPUB	2004/03/02 09:07
-	5	@pd < "20011127" and (EDI same schedule) and database	USPAT; US-PGPUB	2004/03/02 12:00

Introducing XML/EDI...

"the e-Business framework"

Bruce Peat & David Webber



German Version

The XML/EDI framework opens the door to a new paradigm for processing documents and exchanging transactions. In this article we explore how XML/EDI "smart" documents can be used to address some very important issues of communication both internal and external to organizations.

By using standards for accessing components of an XML document and adding style information to XML documents, XML/EDI documents and their included objects can be exchanged, viewed, searched, catalogued, and routed ("pushed"). The XML/EDI framework provides all the fundamental building blocks of Electronic Commerce (EC).

The bottom line, XML/EDI has the potential to change the way organizations manage and transfer their "business" information. While the Internet XML ground swell is only just beginning, by 1998 XML/EDI could well be one of the dominant forces in e-Business. Let's find out how this will be done.

Topics:

About this article

This article is written for those who want to know how XML (e**X**tensible **M**arkup **L**anguage) and EDI (**E**lectronic **D**ata **I**nterchange) provide a catalyst for building E-business solutions. It assumes some technical knowledge of the Internet, Electronic Commerce and EDI. References to these can be found at the XML/EDI Group Web site: http://www.xmledi.org.

The article describes the important concepts of XML/EDI and how you deliver and process simple, durable and effective business transactions via electronic means.

To achieve these goals requires a method that it is not only extensible into the future but also adaptable to incorporate new technologies and requirements. To ensure broad adoption the technology selected needs to be widely and freely available as an open standard.

By selecting a paradigm that by its very nature is dynamically defined, extensible and simple, these goals are intrinsically met.

XML/EDI involves much more than just dropping EDI into an XML wrapper. An XML/EDI framework includes the use a set of complementary and powerful technologies. Our goal is to introduce you to these XML/EDI concepts and what is achievable.

If after reading this article you have further questions, please post them to the XML/EDI Group's mailing list.

What makes XML/EDI work?

XML/EDI is the fusion of five technologies. It is this combination of capabilities that makes XML/EDI so powerful. The five technologies are:

- XML
- EDI
- Templates
- Agents, and
- Repository.

Each component adds unique tools that leverage the other pieces. In the past EDI was very static, today the XML/EDI framework provides is an exciting dynamic process that can be infinitely extended. We now look at each technology to give you an overview, while later sections provide more in depth looks at each.

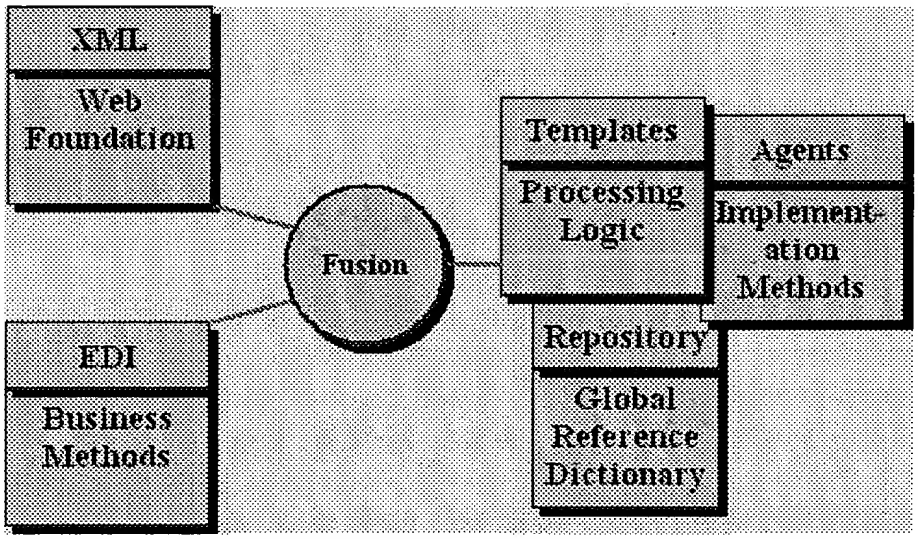


Figure 1. XML/EDI the Fusion of Technologies

The Power of Five:

XML itself provides the foundation. The Web was born on the abilities of the HTML language, itself a very limited subset of the original and highly complex SGML document syntax. Now XML has been created that sits between the two, not as complex as SGML, but vastly more capable than HTML. XML tokens and frameworks are the

syntax that transports the other components across the network. XML tokens replace or supplement existing EDI segment identifiers. XML also brings with it all the rich capabilities and transport layers of the Web and the Internet in general.

EDI is the grandfather of the electronic commerce. The ability to express data in a simple format and send it to someone else so they can interpret the information they have just received. XML/EDI provides 100% backward compatibility to existing EDI transactions, while moving EDI forward to the next generation. This means we do not have to discard the investment in existing EDI systems and knowledge.

Templates, these Rules provide the glue that holds the whole process together. Without them you cannot express in the XML alone all the details of the work that needs to be done. Templates are referenced or travel along inside the XML as a special section and set of tokens, and can be easily read and interpreted, they look rather like a spreadsheet in layout and content and they are supplemented by what XML calls Document Type Definitions (DTD's). DTDs enable transaction interoperability, rules enable processing (which may include presentation) of transactions. DTDs let two organizations understand each other's data. Rules define what happens to that data.

Agents, both interpret the *Templates* to perform the work needed, and also interact with the transaction and the user to create new templates for each new specific task, or look up and attach the right template for existing jobs. They also can reference DTD's to determine display characteristics for forms. This is where Java and ActiveX fit in. Right now they provide the best medium for creating Agents, and of course, XML structure containing these pieces can be referenced or transported to wherever they are required. Initially these agents will be simply driven by the *Templates*, with additional capabilities anticipated to follow later.

Repository - *Shared Internet Dictionaries* are already in use by hybrids of traditional EDI systems such as the BSI which uses a dictionary that allows users to manually look up the meaning and definition of EDI elements. The *Shared Internet Dictionary* concept takes this to the next level, and provides automatic lookups rather like the more advanced Internet search engines now do. This component provides the semantic foundation for business transactions and the underpinning that the software *Agents* need to correctly cross-reference entities. (This concept is also referred to as "repositories" in this article, and includes adding DTD's to the repository as well).

Combining these components together provides a system that delivers information, not just data, and the processing logic that is required.

What will XML/EDI do?

XML/EDI provides four core models of use. These include traditional EDI deployment methods, along with new document-centric capabilities. *Figure 2* shows at a glance the four core models. Further in the document we discuss some example interactions.

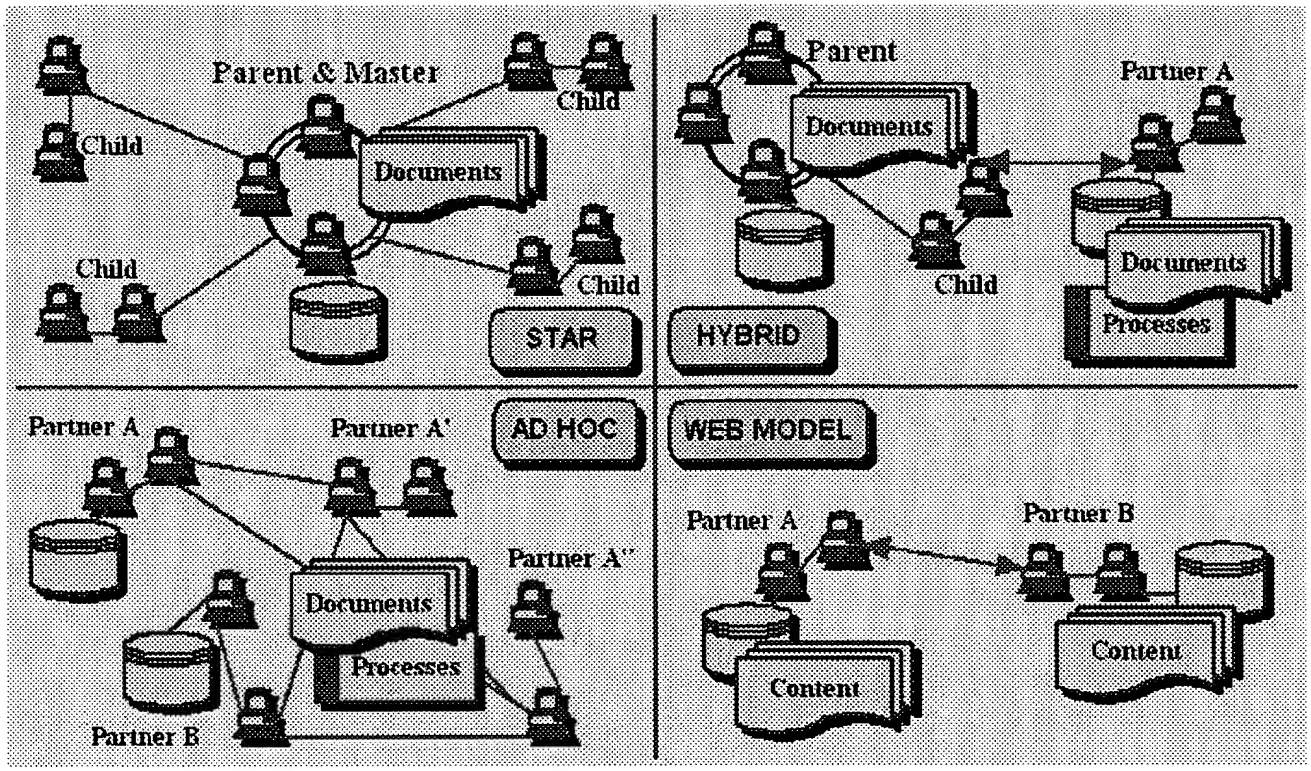


Figure 2 XML/EDI Transaction Models

The next section discusses frameworks that enable these models. The "star" model shown is the classic EDI model, where a major business partner or organization sets the standards for its trading partners. The "ad hoc" model is the new net-based model. Smaller trading partners setup their own ad hoc interactions, these in time may evolve into more formal methods, or they may not. The "hybrid" model is a combination of the first two. Here a "Star" model is extended by trading partners, by creating new versions of frameworks, and by linking in their own ad hoc ones. The "Web" model is a document-centric model. Where content is the most important information being exchanged. Content can either arrive driven by pre-set rules, or be requested, or be broadcast. The classic example for this is an electronic catalog, and the associated "Request For Quotations" (RFQ) dialogs.

Why a framework?

XML/EDI provides for the infrastructure of a wide variety of EC systems; from searchable on-line catalogs to robust machine-to-machine transaction subsystems. The XML/EDI initiative is to provide open solutions for implementing E-business systems today. The operative word here is "solutions". There isn't just one solution for all of the E-business scenarios. Each scenario has its own requirements and goals. This is why we need a framework and not an application or module.

The goal of the framework is to provide formal interfaces for commercial EC components to interoperate. For XML/EDI to be successful these interfaces will be open and yet standardized. The business model is either of ad hoc interactions between small groups, or agreed upon national or international frameworks, such as those by trade associations or industry bodies.

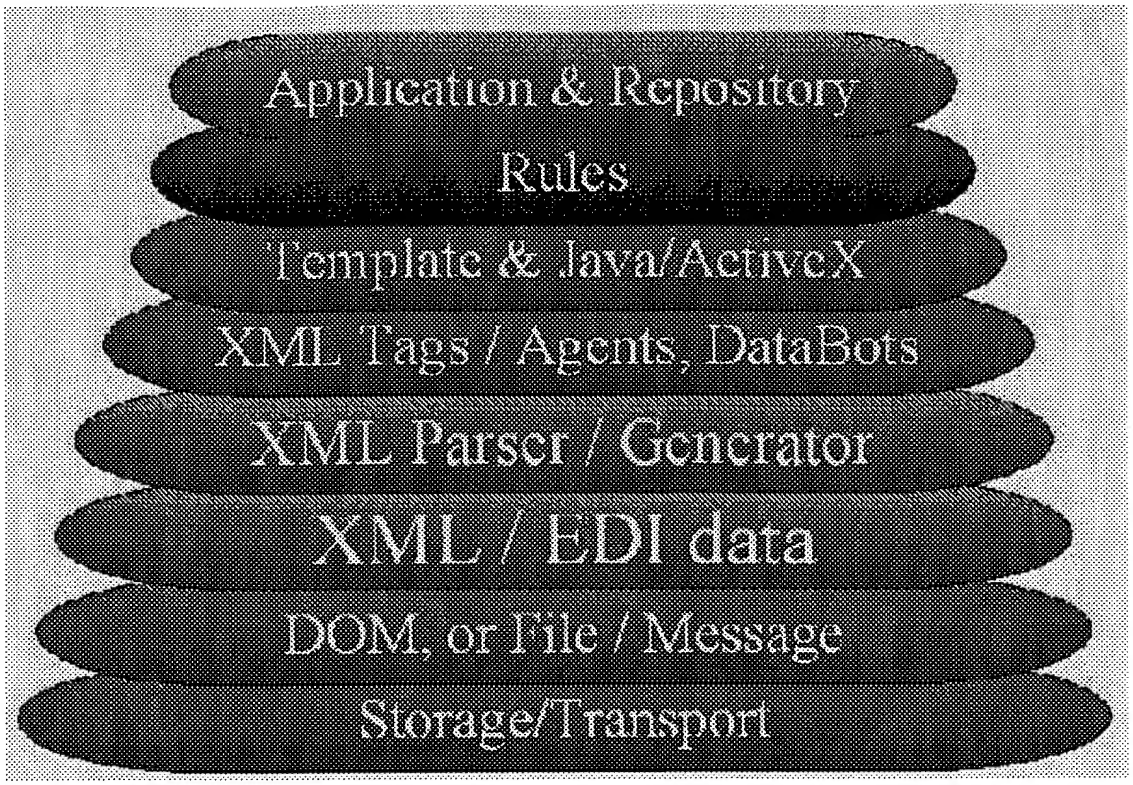


Figure 3 - XML/EDI Layered Architecture

Figure 3 depicts the technical layers that a base XML/EDI system can be built. Not all layers are required to achieve results with XML/EDI. The framework builds upon and enhances the XML and EDI standards and defines the blueprint of EC component interaction. For example, catalog vendors may only want to implement the XML tagging. The tags are defined in "standard" repositories. The various layers support different targeted EC systems. Overall each successive layer provides increasingly more sophisticated capabilities so as to handle more demanding needs.

The framework is not a "all-or-nothing" nothing implementation. Each EC component is designed to be able to be used independently -- interfacing with each other as defined by XML/EDI standards. You can create a DTD for formally define the strucuture of EDI messages without having to use it to validate well-formed XML documents. You can simply place EDIFACT/X12 messages in an XML shell element or you can code the message, or parts of it, in XML. You can chose to formally validate messages, or simply check that they are well-formed XML. You can link your messages to data stored elsewhere, or have it all stored intermally. You can link your documents to rule-templates, or have in-built (or no) rules for processing the data you receive. Its up to you how you mix and match the options.

End Users interact with these technical layers through highly visual desktop tools. Developers use EC components to build these tools. Users interface with their documents as they would use any tool in their office support products - word-processing, spreadsheets, databases or whatever metaphor is most suited to the application.

What makes XML/EDI different?

The main difference with XML/EDI and other mechanisms is that a system can encode the

document's information much more precisely and in a very rich structure than was previously possible with earlier formats. Through the use of XML tags and DTD's, XML/EDI transaction are self-describing--applications processing XML/EDI documents can "understand" a transaction with access only to the content of the transaction.

There are several reasons why we anticipate XML/EDI to be the technology of the future. Some of the reasons are listed below -

XML/EDI...

- is built on open standards
- provides for self-describing transactions (*XML*)
- allows tool vendors to build on existing products
- tools include workflow and document management functions as well as EDI "mappers"
- interfaces with legacy systems very well
- framework uses an evolving best of breed philosophy - i.e. dynamic shared dictionaries
- will allow for object-based documents - data & rules reside together
- provides the organization the path to move down to a document-centric environment
- enabling more flexible business models (*Rules/Agents*)
- cheaper and easier to implement
- access to a greater number of trading partners (*see previous bullet*)
- and most importantly: access to "interactive" transactions enabled by the web rather than being limited to "system" or "batch" transactions.

Another aspect of XML/EDI which needs to be mentioned -- only a few times in computing history has the attention to the collective technologies and their potential solution to solving today's business problems been so focused - there is excitement in the air and a sense of the achievable.

Document-centric environment?

The world has changed from thirty years ago, and now requires a dynamic and vibrant tool that matches the organized yet ad hoc nature presented by both modern business practice and its manifestations in the Internet itself. The Internet is re-writing the rules on how people interact, buy and sell, and exchange goods and services.

Today we see document-based management systems integrating with relational database management systems (RDBMS). If we look at the various offerings with "universal servers"; Oracle, DB2, Informix and "message stores" such as Notes and MS-Exchange these give us our roadmaps for predicting the possible types of business computing systems. These systems are/will combine fielded data with full text-based subsystems - database systems and document-centric tools are supplementing each other and providing to the end users new paradigms for working with documents.

This integration of data-centric and document-centric tools is coming together for larger audiences than that of XML/EDI. For example, it is rumored that Microsoft is considering using XML as Exchange's native file format to speed access to/from the message store. This step is needed if Exchange is to be able to handle 5 million users on a clustered NT environment, 2 or 3 years from now. By building upon these types of XML mechanisms, XML/EDI can leverage these tools, such as workflow, cataloging, routing and searching. For XML/EDI, it is often referred to as, "all this stuff comes free" or "all part of the evolving XML technology".

If our assumption is true, applications will evolve to using the built-in browser functions as their front-ends. Developers won't be writing individual screens to view transactions, i.e. an invoice or patient record, but instead use the integrated browser which will "know" how to adapt for the transaction. The document-centric tools will be entering the workplace more and more, XML/EDI just builds on and leverages these tools.

XML/EDI documents will serve as a structure that contains data, and will include instructions on how the transaction should be processed or displayed. Based on user defined rules the document will be routed in its own workflow process, triggering events on its own. In the simple case, the document will be able to find the application (or user) by using the search, classifying, and routing mechanisms, instead of the applications (or user) having to find it. The document will have transaction status self-contained for applications (or users) to set/interrogate and will know it is one in a set of linked set of documents used in the workflow.

Now lets us pose the question again... What if you had the tools to process the transaction as a document? This is the powerful concept behind the "universal servers" of today, combining the strength of the RDBMS with document management. With a variation on the tools today; i.e. SGML, EDI mappers, we can leverage this technology to store, search, and manipulate our documents. We can use document-centric tools to complement those tools EDI administrators have today *(RDBMS)*.

Isn't XML used only for Web-based documents?

No. The first acceptance of XML is already taking place for the Internet, and for a wide range of Intranet applications that are dependent upon the Browser. But as these applications develop they will feed the tools required for the use of XML/EDI . XML/EDI is not tied to the browser, client based model. XML/EDI delivers true server based batch processing as well. And of course via any transport method -- an XML/EDI document can be exchanged by floppy, LAN, WAN, dial-up modem, Internet, etc.

Where is the value-add of XML/EDI?

One of the XML/EDI objectives is to makes the transfer of documents between organizations as easy as possible to the point that it is transparent to the user. This will allow for workflow or "pipelines" to extend across organizations. As a subset, this includes software agents to reach out on to the Internet to read from and "make sense" of online catalogs. The workflow of documents should be able to route to a trading partner as easily as a colleague down the hall. This transparency is achievable with XML/EDI.

What studies have shown is that only a few companies who have implemented EDI have leveraged all of the benefits that are provided when automating the data interchange process. Most companies use EDI only as a transport mechanism to send fixed structured data from and to trading partners. Some organizations even re-key the received EDI data into their other systems! EDI usually is imposed upon corporations by their partners, and EDI is seen only as a task to keep their customer base happy. Very few corporations begin to get involved with EDI by saying, "EDI is a good thing and think I will save my company money, by cutting processing costs." EDI usually happens in a company because of a partner has dictated its use, so no wonder EDI is left at the organization's doorstep.

The difference between the above EDI efforts of the past and XML/EDI is primarily due to changes

in the workplace. These changes will help with integrating data and processes between organizations. The time will come when users of XML documents will expect their systems to be able to transparently pass on a transaction to a trading partner as easily as just printing it. The short of it, with XML/EDI the transaction does not stop at the gates of the organization, but extends into an organization enabling all facets.

Using XML structures for XML/EDI?

An EDI application for XML provides the structural complexity that supports and parallels today's EDI transaction sets. XML provides a rich document structure that can be nested to any level of complexity. With XML, our documents are like chameleons, capable of being processed by different components, delivered by different mechanisms, and displayed to the user in different ways.

By using the XML extensible tag set, EDI "objects" can be either passed or dynamically reference to objects stored in repositories. The XML/EDI Group is proposing the use of XML as a "carrier" for the document information so that the transaction can carry not only data (like trad-edi), but also code (at each level in the transaction tree). With an element having data properties and code methods, this allows the business elements to be manipulated as "objects". [Note: object inheritance is slated to be incorporated in a later version of XML]. In XML, each document is an object and each element of the document is an object.

The logical structure of the document and tag set can be specified in a Document Type Definition or DTD. (The best-known example of a DTD is HTML, which is defined by a DTD describing the structure of HTML documents.) In a DTD, sets of elements and their attributes are defined; the names that are used as tags are assigned; and the element relationships or transaction is defined. If a DTD is used then programs can validate the transaction's structure. You can validate the structure of an XML/EDI document automatically. If this sounds complex, it is not. Defining your own markup language (DTD) with XML is actually surprisingly simple.

Most workflow systems today; such as Lotus Notes, cannot even begin to store adequately an X12 or EDIFACT transactions as a relational view. Today these systems need to "flatten" out the database or to explain it a different way. Typical workflow documents can only store a single one-dimensional view of the transaction. XML/EDI structure will allow products like Notes the ability to store and process EDI transactions in a relational manner.

Once defined, rules can be applied to the objects in the transaction. We will get to this later.

Dynamically referenced objects?

An important note: the data and code do not have to reside in the same transaction! How can this be?

The code and data can be externally referenced - via links to dynamic reference(s), i.e. URLs. XML links can be between two or more resources, and resources can be either files (and not necessarily XML or HTML files) or elements in files. Links can be between more than one resource, they can be specified outside the actual documents themselves and the linked-to element inside a resource. The link can be specified in very powerful ways. The element can be identified with an attribute, a position in the element structure, or one can even specify that the link goes to things like the "ship to" address for the third item in a purchase order. This XML-Link makes XML/EDI very flexible.

The "standards" or dictionaries are stored in a repository(s) for dynamic reference. No more shoe-horning data into improper codes or wild use of ZZ (codes not found in the dictionary) -- just add the new code and its meaning to the repository! XML/EDI framework describes the use of the repositories; interface and replication. XML is a language, the repositories allow for common EC definitions among organizations in a dynamic fashion -- the repositories provide us with a richer language.

How are rules applied to objects?

The answer lies in the Document Object Model (DOM) as defined by the W3C DOM Workinggroup. (Note: Microsoft recently has introduced DOM or Dynamic HTML into IE 4.0). The Document Object Model makes it possible to address all the elements of an HTML document. In effect, each element is a programmable object. By using the Document Object Model, XML/EDI documents are able to combine the content, the rules that controls the transaction, and the view in one file/transaction. XAPI-J provides the standards for Java components to address and navigate the objects in the DOM tree. These objects are our business objects as defined in the DTD. At the user level, rules are graphically presented very much like setting up the "assistant" templates in your E-Mail program where you move messages from one folder to another depending on the "Sender" or "Subject" message.

Software Agents Explained

In this section we answer the question *"isn't software agents all research and academia with no real products?"*.

First one must appreciate that there are a range of software agents. The term encompasses from the very simple to the very complex. In the initial XML/EDI framework described here, the use of agents is either simple, or using already proven agent methods and technologies. Some examples will illustrate the points here. One agent component is required to provide access to users existing databases, or create databases if directed. Java already has these capabilities in the JDBC database component, and also the Microsoft ODBC equivalent. So an agent merely includes these existing capabilities. Another agent may simply provide the ability to help the end user initially configure a particular component when it is first used. This is the familiar "Wizard" style prompting agent that has a simple set of rules and user queries that it processes and creates a set of results.

In the long term also, the more sophisticated agents will be running not at the client Web Browser but on servers on the Internet. Components on the Web Browser will then query these remote agents to obtain results. An example of this is the Global Dictionary Lookup. A local agent receives a type of EDI data segment that it has not encountered before, therefore it will query the agent at the server to classify it and send it back the results.

Another objection to using agents is centered around the security risk. "These agents are Java based and how do I know they are safe?". Obviously in any EDI interaction, the trading partners must predefine who they are allowing to interface with their system. They should also be able to define who they will accept agents from. This will add greater security, since obviously a rogue agent introduced into a global network would be propagated widely. These are issues however that are not unique to XML/EDI, and so these are being solved in a wider arena. XML/EDI however needs to ensure provisions are designed into the system to take advantage of such security methods.

Of course the benefit of using agents is to make the system fault tolerant and also much easier to use. In many cases agents will resolve problems without the user being aware there was a problem in the first place. The example of the Global Dictionary lookup just given being one such case. Once the agent receives the information about the newly encountered data segment, it is then able to correctly process that new element. Perhaps the new segment relates to a changed rule for processing an existing segment. This existing segment value can now be correctly validated using the new segment, and so on.

How about searching and routing transactions?

XML/EDI will be able to use the many search tools that will/are being adapted for XML. The infrastructure will allow for the searching of business objects in addition to keywords. The business objects in XML documents will allow for more intelligent searching than with full-text search engines today. There are already SGML query languages that are similar to SQL in power. Expect these to be integrated with relational database fielded searching; ConText from Oracle and the Monarch project from Microsoft using OLEDB. With standardized DTDs for different applications one could retrieve information much more accurately than today. The relationships in the structures can be used as well as the objects themselves in the query. The DTD allows for precision relational searches of the XML/EDI documents either in your message store, on the Web, i.e. catalogs, E-Mail subsystems, and so forth.

Once searched, the transactions can be classified and routed based on application-type information. Much work has taken place and continues to take place in this area of product development such as Intelliserv from Verity. Routing and status information is stored in the document itself. Routing can be as simple as vectoring a document to a URL. This is how the Web works today -- we push and pull HTML pages to and from servers on a click of a mouse on the Send button. This means, we can have distributed applications by simply using the infrastructure present on the Internet today. This distributed architecture aids us when mapping to legacy applications as discussed later in this article. In addition, agent software has exploded in parallel with the Internet. The XML/EDI framework utilizes these agent components to make the user's task simpler.

What about displaying the transaction?

Now what was that about person-to-application, Web technology? Oh, yes, as an added benefit, the leading browsers will be supporting XML, allowing for the documents/transactions to be viewed exactly the way the user wants it, anyway they want it. If users want the transaction to look like their paper form, it will. Because the presentation can be stored in the document, authors have full control of the view during page makeup.

Even better we don't have to create a template or form to view the transaction. As the document passes from department to department or trading partner to department, it gets viewed appropriately - and automatically. No need for endless man years of programming effort maintaining templates for specific transactions. An XML API with Java (XAPI-J) to be supported by all XML processors (browsers and other XML/EDI tools). This API will make it possible to have Java applets that can be used to change the display of XML/EDI-encoded information in Web browsers. The way the transaction is viewed can be dynamic; simply setup by its author once using a XML/EDI editor for the transaction. Each node on the documents workflow or pipeline redisplays the document as specified by the author.

There are other benefits to storing the structured presentation with the structured data.

- aided by labels, searches are simpler
- archiving of the document becomes simpler
- page makeup while the document is being retrieved from the message store becomes much simpler
- even reading and navigating the documents is simpler

The overall result is that XML/EDI documents are simpler to use. Even adding a "yellow sticky note" to a document is no problem with XML/EDI.

Interfacing to a legacy system?

or exchanging information between disparate systems

Today, companies use EDI only to transport fixed data from here to there; extracting the data out of their database fields, formatting it, sending it, and then the trading partner receives it, takes it apart and tries to figure where to put it in their database fields. This is a very common example of a legacy system. Let's say we want to extract data from an application into our workflow or we have just routed the document to the application's URL, and now we need to export/import the information from/into the application. We need to translate our XML/EDI document out of/into our RDBMS. What tools do we have with XML/EDI to assist with this?

Actually extracting the data to be sent is a somewhat easier problem than deciphering upon receiving an inbound transaction due to the fact that they are self describing. Our rules based templates do just fine, and aren't much different than that of today's mechanisms. Because a DTD gives a standard format for information related to a specific subject it can be used to simplify the exchange of information between different sources. Many kinds of applications have or will have standard DTDs. This means that systems can use these common base DTDs to exchange information with each other, regardless of their internal format. Using DTDs in this manner is one of the most important reasons why XML/EDI will enable all organizations large, medium or small to exchange data.

Where XML/EDI shines is with handling the harder inbound documents, with traditional EDI the data is all we get from our trading partners. Remember with XML/EDI we get not only data, but any rules; application logic, required by the system we are interfacing. The XML/EDI advantage comes from the fact that we aren't just passing data elements, but passing business objects. If we apply these objects and their EC mechanisms to our translator problem, we untie our hands from data only inferences, allowing us to apply agent technology to assist in the mapping challenge.

As mentioned above, with a distributed architecture via the use of URLs, mapping can take place at the various applications. This is in lieu of forcing the centralized server approach. A distributed architecture aids us when mapping to legacy applications because we can take advantage of the knowledge stored within the application. Validation, cross-reference look-up tables, and most importantly application logic can be used during the mapping process. Each user effectively becomes a trading partner, sending and receiving transactions, internal and external to the organization.

Example...

To understand the set of interactions here, let us consider the simple example of two businesses each with the need to send information to the other. Partner A starts the XML/EDI software, selects the databases from his business application, reviews the list of information presented and chooses the

data to be sent. The XML/EDI software references the Global Dictionary, it looks for an existing template with significant matching, and also the occurrence of the same business elements in the dictionary. Matches are found for the obvious items, such as postal code and invoice date, but definitions are taken from the database field formats for the remainder. In this case no suitable template is found, the system creates one that describes the information being sent, including its record structures.

The system also creates an XML document, displays this in a Web browser, and allows Partner A to both modify this data view and select the information required to be sent. Now the information is ready it is sent to Partner B.

Partner B receives the information and she repeats the process, this is the first time she has received this template and data format, so the receiving XML/EDI software queries for an action. It uses the incoming template to help match the information to the receiving databases of Partner B's business applications. Partner B now reviews the data and matches it to the same locations and items in the receiving databases. Agent software guides this process by making the obvious matches for her. The XML/EDI software agents now create a second template and rules that correspond to this set of data transformations. The Agent software is also able to automatically resolve data structure transfers and create the necessary data records in the receiving system. Partner B reviews the received data in her Web Browser and adjusts the view. Partner B can now use her template to also extract information from her system and send it back to Partner A. Partner A now has the original template, and the new template to enable reversing the process to receive this new information. Now in this example we illustrated a simple transfer. Obviously Java applets or ActiveX components can be exchanged to calculate complex interactions and elements. An example would be the interest payments on past due amounts, given five fields, the amount owed, the date the payment was due, and the date the amount was paid, and the amount that was paid, and the interest rate.

One further note, in this example, the interaction was driven through a Web browser interface. However once the templates and formats are created, these can equally easily be used in a batch style exchange. In this case a batch process determines the data records to be sent at that time, creates the XML/EDI documents and sends these directly, either to a batch process at the receiving end, or to a Web Browser interface, or even an E-Mail system for delivery and onward processing. This begins to illustrate the power and flexibility of the XML/EDI system.

We have been searching for the ultimate solution for exchanging information between disparate systems for many years - to date XML/EDI comes closest to our ideal solution.

XML/EDI for existing EDI.

One of the most powerful capabilities provided by the Rule Template component of the XML/EDI "Power of Five" is the ability to define templates that map data to and from traditional EDI messages (such as X12, EDIFACT, or HL7). This means that XML/EDI can seamlessly interface with existing EDI systems and provide 100% backward compatibility. The template contains the structure of either the incoming or outgoing EDI message formats, and this allows XML/EDI to exactly re-create them without needing an external translation process. Added to this is the ability of XML to define a presentation definition in the DTD's and XML structure that instructs the Web Browser how to display and edit the data content (if this functionality is required). Therefore an existing EDI message can be read in and the details displayed to the user directly. This will actually allow traditional EDI systems to reach out to users they have hitherto been unable to reach because of cost factors in deploying and maintaining traditional EDI translator systems at remote sites.

XML/EDI f r the Power Hungry?

Critics scorn at an increase in size of a transaction due to the increase in tagging characters. A common assertion is that XML/EDI sounds expensive in terms of processing and bandwidth. The critics are correct, there will be additional overhead in the size of a transaction. The number of characters could be as high as 50% more than a traditional EDI transaction today. We believe bandwidth of the communications infrastructure is not a constraint for electronic commerce today. The increase in tagging characters is a small price to pay for the many benefits gained.

Consider a current EDI system where everything has been hand tuned to minimize message size, and it is also processing 200,000 messages a day. Surely this Web XML/EDI based stuff cannot handle that with all of its token overhead?

This is where one has to realize the flexibility of the method. Certainly the preferred approach is to simply add all the details and structure you need, because in low volume Web interactions this is the best way. Ensuring the receiver can fully process the information passed.

However, because the template and XML structure information are separate components, one can easily take the opposite approach. Instead of using extensive meaningful tokens, one can instead select two byte descriptors as tokens that are machine readable only, and send only the data section of the XML message. Now your semantic overhead is comparable to traditional EDI.

The templates and Java applets are sent only once as a separate exchange. Once they are invoked the templates contain the logic to decode the cryptic tokens into the full length versions, either by referencing the global dictionary, or simply having the expansions within the templates themselves. So hexadecimal "" corresponds to "Last Invoice Billing Date", and so on. Similarly you could even compress the message content itself in line, and then use a Java applet with the expand routine to retrieve the actual data. This is XML/EDI for the power hungry.

What is the existing alternative?

Compare using XML/EDI as described above with that of your current processes. Even the simpler HTML Web interface, which thousands of corporations are developing today seems archaic when compared to the advantages afforded XML/EDI.

To contrast the differences, an HTML form is either created or dynamically generated with text, drop-down edit fields, etc. to house information to be captured from a user. Hard coded JavaScript or Java objects reside on the page (not attached to a business object) and responds to these generic events to provide validation. Upon submission to a server via hard coded CGI, NSAPI, ISAPI, or ASP (Active Server Pages) syntax these can then be stored to a proprietary database or mapped onto an X12 or EDIFACT transaction set. And if the Web page is going to an external company... no this article doesn't cover the problems on the receiving end of a legacy system. Suffice to say we have the same "mapping" problem between companies. Does this HTML based approach work for some applications? Yes, but very limited compared to the options afforded with XML/EDI.

XML/EDI provides an extendible tool that requires less physical coding to develop and deploy.

Summary

The framework is best summarized by stating the noting the following:

XML/EDI in short...

- Self-describing transactions
- Enables more flexible business models
- Cheaper and easier to implement
- Access to a far wider number of trading partners
- Security, reliability, and robustness
- Interactive transactions enabled by the Web, not just "system" or "batch" transactions.

XML/EDI in detail...

1. is based on industry standards
2. allows for presentation, application logic, and data to be contained in one document
3. is an evolution rather than a revolutionary approach, it builds on current Web technologies and yet also interfaces with legacy EDI systems
4. provides for common tagging for accurate interpretation of documents
5. is a unified structure for batch, interactive, forms-based, and real-time exchanges
6. allows for the use of document-centric tools in addition to database facilities to manipulate, store, search transactions

"... with XML/EDI the process does NOT stop at the gates of the organization, but extends into an organization, enabling all facets."